AN ANOMALY-BASED WEB APPLICATION FIREWALL

Carmen Torrano-Gimenez, Alejandro Perez-Villegas and Gonzalo Alvarez
Instituto de Física Aplicada, Consejo Superior de Investigaciones Científicas, Serrano 144 - 28006, Madrid, Spain

Keywords: Web attacks, Anomaly intrusion detection, Web application firewall.

Abstract: A simple and effective web application firewall is presented. This system can detect both known and unknown web attacks following a positive security model. For attack detection, the system relies on an XML file, which thoroughly describes normal web application behavior. Any irregular behavior is flagged as intrusive. An initial training phase is required to statistically characterize how normal traffic for a given target application looks like. The system has been tested with a real web application as target and an artificial request generator as input. Experiments show that after the training phase, when the XML file is correctly configured, good results are obtained, with a very high detection rate and a very low false alarm rate.

1 INTRODUCTION

Web applications are becoming increasingly popular and complex in all sorts of environments, ranging from e-commerce applications, to social networks, to banking. As more vital information and services are moved to web platforms, their interest as potential targets increases. As a consequence, web applications are subject to all sort of attacks, many of which might be devastating (Alvarez and Petrovic, 2003). Unfortunately, conventional firewalls, operating at network and transport layers, are usually not enough to protect against web-specific attacks. To be really effective, detection is to be moved to the application layer.

Intrusion detection is the act of detecting malicious actions and behaviors that can compromise the security and trust of a computer system. An Intrusion Detection System (IDS) analyzes information from a computer or a network to identify possible security breaches. Traditionally, IDS's have been classified as either signature detection systems or anomaly detection systems. The first method looks for signatures of known attacks using pattern matching techniques against a frequently updated database of attack signatures. The second method looks for anomalous system activity: once normal behavior is well defined, irregular behavior will be tagged as intrusive. An hybrid intrusion detection system combines the techniques of the two approaches.

Signature-based IDS's rely on large signature databases and are unable to detect new attacks. To work properly, databases must be updated frequently and signature matching usually requires high computational effort. Anomaly-based IDS's overcome these problems. However, in rather complex environments, obtaining an up-to-date and feasible picture of what "normal" network traffic should look like proves to be a hard problem.

The results of signature-based WAFs depend on the actual signature configuration for each web application, and cannot be compared with anomaly-based WAFs.

Varied techniques have been used to solve the general intrusion detection problem (Patcha and Park, 2007) , like clustering (Petrović et al., 2006). However some of these techniques are not appropriate to solve the web intrusion detection problem due to the difficulty of coding a request as an input. Some of the main works developed to solve web attack detection are (Kruegel et al., 2005) which follows an statistical approach and (Estévez-Tapiador et al., 2004) which uses Markov chains.

In this paper, a simple and effective anomaly-based Web Application Firewall (WAF) is presented. This system relies on an XML file to describe what a normal web application is. Any irregular behavior is flagged as intrusive. The XML file must be tailored for every target application to be protected.

The rest of the paper is organized as follows. In Sec. 2, a system overview is given, where system architecture, normal behavior modeling, and attack detection are explained. Section 3 refers to experiments. Traffic generation, the training phase, the test phase and results are also described. Section 4 describes system limitations and suggests future work. Finally, in Sec. 5, the conclusions of this work are captured.

2 SYSTEM OVERVIEW

2.1 Architecture

Our anomaly-based detection approach analyzes HTTP requests sent by a client browser trying to get access to a web server. The analysis takes place exclusively at the application layer, specifically HTTP protocol. Thus, the system can be seen as an anomaly-based Web Application Firewall, in contrast with existing signature-based WAFs (ModSecurity, 2009). By analyzing HTTP requests, the system decides whether they are suspicious or not. An HTTP request is considered suspicious when it differs from the normal behavior according to some specific criteria.

In our architecture, the system operates as a proxy located between the client and the web server. Likewise, the system might be embedded as a module within the server. However, the first approach enjoys the advantage of being independent of the web platform.

This proxy analyzes all the traffic sent by the client. The input of the detection process consists of a collection of HTTP requests $\{r_1, r_2, \ldots r_n\}$. The output is a single bit a_i for each input request r_i, which indicates whether the request is normal or anomalous. Since the main goal of our system is to detect web attacks trying to reach the web server, a single bit alert and event log upon detection is enough. However, it could be more practical to give the system the ability to operate as a firewall, that is, allow normal requests to reach the server and block malicious ones. Therefore, the proxy is able to work in two different modes of operation: as IDS and as firewall.

In detection mode, the proxy simply analyzes the incoming packets and tries to find suspicious patterns. If a suspicious request is detected, the proxy launches an alert; otherwise, it remains inactive. In any case, the request will reach the web server. When operating in detection mode, attacks could succeed, whereas false positives don't limit the system functionality.

In firewall mode, the proxy receives requests from client users and analyzes them. If the request is valid, the proxy routes it to the server, and sends back the received response to the client. If not, the proxy blocks the request, and sends back a generic denegation access page to the client. Thus, the communication between proxy and server is established only when the request is valid.

A diagram of the WAF's architecture is shown in Fig. 1.

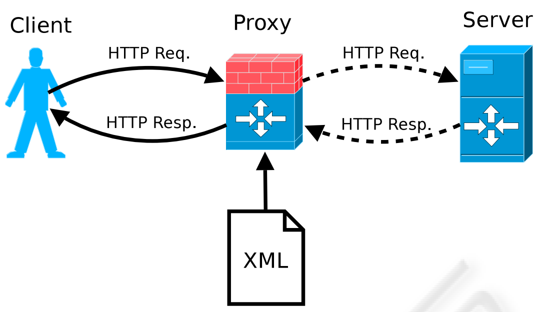

Figure 1: Web Application Firewall Architecture.

2.2 Normal Behavior Description

Prior to the detection process, the system needs a precise picture of what the normal behavior is in a specific web application. For this purpose, our system relies on an XML file which contains a thorough description of the web application's normal behavior. Once a request is received, the system compares it with the normal behavior model. If the difference exceeds a given threshold, then the request is flagged as an attack and an alert is launched.

The XML file consists of a set of rules describing the different aspects of the normal request. More precisely, the XML contains rules regarding to the correctness of HTTP verbs, HTTP headers, accessed resources (files), arguments, and values for the arguments. This file contains three main nodes:

Verbs

The *verbs* node simply specifies the list of allowed HTTP verbs. Requests using any other verb will be rejected.

Headers

The *headers* node specifies the list of allowed HTTP headers and their allowed values.

Directories

The *directories* node has a tree-like structure, in close correspondence to the web application's directory structure.

1. Each directory in the web application space is represented in the XML file by a *directory* node, allowing nesting of directories within directories. The attribute *name* defines these nodes.

2. Each file in the web application space is represented by a *file* node within a *directory* node and is defined by its attribute *name*.
3. Input arguments are represented by *argument* nodes within the corresponding *file* node. Each argument is defined by the following set of attributes:
   - *name*: the name of the argument, as expected by the web page.
   - *requiredField*: a boolean value indicating whether the request should be rejected if the argument is missing.
4. Legal values for arguments should meet some statistical rules, which are represented by a *stats* node within the corresponding *argument* node. These statistical properties together give a description of the expected values. Requests with argument values exceeding their corresponding normal properties by a given threshold will be rejected. Each relevant property is defined by an attribute within the *stats* node. In our approach we considered the following relevant properties:
   - *Special*: set of special characters (no letters and no digits) allowed.
   - *lengthMean*: mean value of input length.
   - *lengthDev*: std deviation of character length.
   - *letterMean*: mean number of letters.
   - *letterDev*: std deviation of number of letters.
   - *digitMean*: mean number of digits.
   - *digitDev*: std deviation of number of digits.
   - *specialMean*: mean number of special characters (of those included in the propertie Special).
   - *specialDev*: std deviation of number of special characters (of those included in the propertie Special).

   When an incoming request is received, the features of each argument value are measured. Since these measurements are quantitative, it is possible to compute the distance to the average value. If the distance is greater than the corresponding threshold (scaled by the standard deviation), the value is considered anomalous.

The construction of the XML file and the adequate selection of the threshold parameters are crucial for a good detection process. An example of XML configuration file is shown in Fig. 2.

```
<configuration>
<verbs>
  <verb>GET</verb>
  <verb>POST</verb>
</verbs>

<headers>
  <rule name="Accept-Charset"
        value="ISO-8859-1"/>
  <rule name="Accept-Charset"
        value="utf-8"/>
</headers>

<directories>
  <directory name="shop">
    <file name="index.jsp"/>
    <directory name="public">
      <file name="add.jsp">
        <argument name="quantity"
                  requiredField="true">
          <stats
            Special=""
            digitDev="0.0"
            digitMean="100.0"
            lengthDev="0.1410730099"
            lengthMean="1.93"
            letterDev="0.0"
            letterMean="0.0"
            otherDev="0.0"
            otherMean="0.0"/>
        </argument>
        ...
```

Figure 2: XML file example.

2.3 Detection Process

In the detection process, our system follows an approach of the form “deny everything unless explicitly allowed”, also known as *positive* security model. This approach is always more secure than looking for attack signatures in incoming requests, known as *negative* security model. However, the positive approach is prone to more false positives, with a possible impact on functionality and user comfort.

The detection process takes place in the proxy. It consists of several steps, each constituting a different line of defense, in which a single part of the request is checked with the aid of the XML file. If an incoming request fails to pass one of these lines of defense, an attack is assumed: a customizable error page is returned to the user and the request is logged for further inspection. It is important to stress that these requests will never reach the web server when operating in firewall mode.

The following are the lines of defense considered in our system.

- **HTTP Verbs**. The system can be configured to accept a subset of HTTP verbs. For example, in the applications in which only GET, POST and HEAD are required to work correctly, the XML file could be configured accordingly, thus rejecting requests that use any other verb.

- **HTTP Headers**. Allowed HTTP headers and their expected data types can also be configured in the XML file, thus preventing attacks embedded in these elements.
- **Static Files**. In this line of defense, the system checks whether the requested resource is valid. For this purpose, the XML configuration file contains a complete list of all files that are allowed to be served. If the requested resource is not present in the list, a web attack is assumed.
- **Dynamic Files**. If an allowed resource is requested, then it is checked whether it accepts input arguments. In this case, the incoming data are checked against the validation rules. These rules include all arguments that are allowed for the resource, and which ones are mandatory in the request. Again, these rules are of the form "deny everything unless explicitly allowed". Thus, if the user-supplied request contains incorrect arguments for an allowed resource, an attack is assumed and the request will not reach the web server.
- **Argument Values**. If an allowed resource with allowed parameters is requested, the value of the arguments is checked. An incoming request will be allowed if all parameter values are identified as normal. Argument values are decoded before being checked. As described in Sec. 2.2, for each resource and parameter, the XML file describes statistical features of normal values. By analyzing actual values, the system decides whether they are anomalous (rejecting the incoming request) or normal (allowing the request).

3 EXPERIMENTS

3.1 Case Study: Web Shopping

The WAF has been configured to protect a specific web application, consisting of an e-commerce web store, where users can register and buy products using a shopping cart.

3.2 XML File Generation

As already stated, the XML file describes the normal behavior of the web application. Therefore, to train the system and configure this file, only normal and non-malicious traffic to the target web application is required. Nevertheless, how to obtain only normal traffic may not be an easy task. Since a statistical approach was used for the characterization of normal argument values, thousands of requests are needed. There are some alternatives to obtain normal traffic:

- Thousands of legitimate and non-malicious users can surf the target web application and generate normal traffic. However, getting thousands of people to surf the web might not be an easy task.
- The application can be published in the Internet, but unfortunately attacks would be mixed with normal traffic. Classifying normal and anomalous traffic is unviable. In (Kruegel et al., 2005) this approach is used. However their training data include attacks, so some attacks cannot be detected, as they are considered as normal traffic.
- Traffic can be generated artificially. Although the traffic is not real, we can be sure that only normal traffic is included.

Normal traffic acquisition is a general problem in attack detection, still to be completely solved. For our purposes, we considered artificial traffic generation to be the most suitable approach.

3.3 Artificial Traffic Generation

In our approach, normal and anomalous request databases are generated artificially with the help of dictionaries.

3.3.1 Dictionaries

Dictionaries are data files which contain real data to fill the different arguments used in the target application. Names, surnames, addresses, etc., are examples of dictionaries used.

A set of dictionaries containing only allowed values is used to generate the normal request database. A different set of dictionaries is used to generate the anomalous request database. The latter dictionaries contain both known attacks and illegal values with no malicious intention.

3.3.2 Normal Traffic Generation

Allowed HTTP requests are generated for each page in the web application. If the page presents a form, the fields are filled out only with legal values. Arguments and cookies in the page, if any, are also filled out with allowed values. Depending on the case, the values can be chosen randomly or obtained from the normal dictionaries. The result is a normal request database (*NormalDB*), which will be used both in the training and test phase.

3.3.3 Anomalous Traffic Generation

Illegal HTTP requests are generated with the help of anomalous dictionaries. There are three types of anomalous requests:

- **Static attacks** fabricate the resource requested. These requests include obsolete files, session id in URL rewrite, directory browsing, configuration files, and default files.
- **Dynamic attacks** modify valid request arguments: SQL injection, CRLF injection, cross-site scripting, server side includes, buffer overflows, etc.
- **Unintentional illegal requests**. These requests should also be rejected even though they do not have malicious intention.

The result is an anomalous request database (*AnomalousDB*), which will be used only in the test phase.

3.4 Training Phase

During the training phase, the system learns the web application normal behavior. The aim is to obtain the XML file required for the detection process. In this phase only requests from NormalDB are used. In the construction of the XML file, different HTTP aspects must be taken into account.

- Argument values are characterized by extracting statistical features from the requests.
- Verbs, headers and resources found in the requests are included directly in the XML file as allowed requests.

3.5 Test Phase

During the test phase, depicted in Fig. 3, the proxy accepts requests from both databases, NormalDB and AnomalousDB, and relies on the XML file to decide whether the requests are normal or anomalous.

The performance of the detector is then measured by Receiver Operating Characteristic (ROC) curves (Provost et al., 1998). A ROC curve plots the attack detection rate (true positives, TP) against the false alarm rate (false positives, FP).

$$DetectionRate = \frac{TP}{TP+FN} \tag{1}$$

$$FalseAlarmRate = \frac{FP}{FP+TN} \tag{2}$$

Once the system has characterized how normal requests look like, the proxy analyzes a set of requests (normal and anomalous) varying a certain parameter. The results are measured by the ROC curves.

A sensibility parameter s can be tuned to obtain different points in the ROC curve. An argument is considered normal only if all its statistical properties are inside the corresponding interval. Otherwise, the request is classified as anomalous and a log is generated.

The three allowed intervals of the argument values are calculated from the statistical features registered in the XML file and the sensibility parameter s:

$$[\mu - \sigma * s, \qquad \mu + \sigma * s], \tag{3}$$

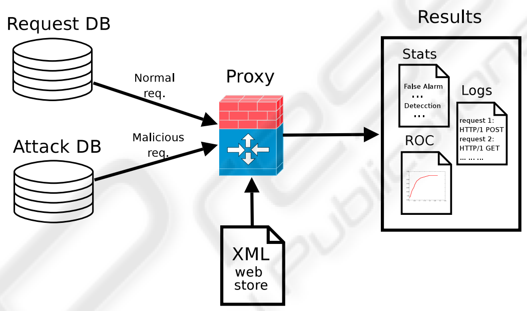

Figure 3: System Test Phase.

3.6 Results

Regarding normal traffic, 36,000 normal requests are generated in 1,000 iterations. With respect to anomalous traffic, approximately 25,000 malicious requests are generated in 500 iterations. Tests have been done varying the sensibility parameter, showing that results only vary when the sensibility parameter values are between 1 and 18. The ROC curve obtained is shown in Fig. 4.

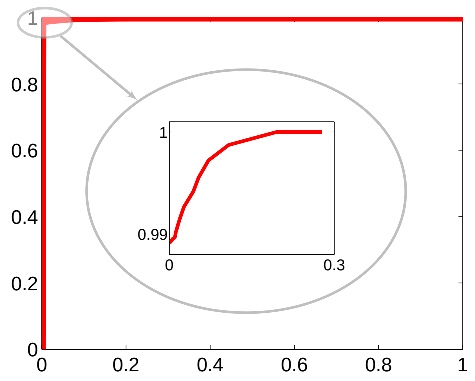

Figure 4: ROC curve of WAF protecting the web store.

As can be seen, very good results are obtained: the number of false positives is close to 0 whereas the detection rate is close to 1. The reason why such good results are obtained is that the XML file closely characterizes the web application normal behavior in an effective manner.

4 LIMITATIONS AND FUTURE WORK

As shown in the previous section, when the XML file is configured correctly, the system succeeds in detecting any kind of web attacks. Thus, the main issue is how to automatically configure the XML description file. In our approach, the XML file is built from a large set of allowed requests in the target web application. However, obtaining only normal and non-malicious traffic may not be an easy task, as was discussed in Sec. 3.2. Therefore, the main limitation consists in correctly implementing the training phase for any web application.

Other limitations arise when protecting complex web applications. For instance, web sites that create and remove pages dynamically, generate new URLs to access resources, or allow users for updating contents, may difficult the XML file configuration. Further modifications of the system will attempt to solve these problems, by statistically characterizing the URLs of allowed resources.

Future work refers to signing cookies and hidden fields in order to avoid cookie poisoning and hidden field manipulation attacks. Also, URL patterns will be used in describing sites with dynamic resources.

5 CONCLUSIONS

We presented a simple and efficient web attack detection system or Web Application Firewall (WAF). As the system is based on the anomaly-based methodology (positive security model), it proved to be able to protect web applications from both known and unknown attacks. The system analyzes input requests and decides whether they are anomalous or not. For the decision, the WAF relies on an XML file which specifies web application normal behavior. The experiments show that as long as the XML file correctly defines normality for a given target application, near perfect results are obtained. Thus, the main challenge is how to create an accurate XML file in a fully automated manner for any web application. We show that inasmuch great amounts of normal (non-malicious) traffic are available for the target application, this automatic configuration is possible using a statistical characterization of the input traffic.

ACKNOWLEDGEMENTS

We would like to thank the Ministerio de Industria, Turismo y Comercio, project SEGUR@ (CENIT2007-2010), project HESPERIA (CENIT2006-2009), the Ministerio de Ciencia e Innovacion, project CUCO (MTM2008-02194), and the Spanish National Research Council (CSIC), programme JAE/I3P.

REFERENCES

Alvarez, G. and Petrovic, S. (2003). A new taxonomy of web attacks suitable for efficient encoding. *Computers and Security*, 22(5):453–449.

Estévez-Tapiador, J., García-Teodoro, P., and Díaz-Verdejo, J. (2004). Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2):175–193.

Kruegel, C., Vigna, G., and Robertson, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738.

ModSecurity (2009). Open source signature-based web application firewall, http://www.modsecurity.org.

Patcha, A. and Park, J. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470.

Petrović, S., Álvarez, G., Orfila, A., and Carbó, J. (2006). Labelling clusters in an intrusion detection system using a combination of clustering evaluation techniques. In *Proceedings of the 39th Hawaii International Conference on System Sciences*, Kauai, Hawaii (USA). IEEE Computer Society Press. 8 pages (CD ROM).

Provost, F., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning*, San Mateo, CA. Morgan Kaufmann.